APPLICATION FOR A UNITED STATES NON-PROVISIONAL PATENT

# AUTOMATED PERMUTATION METHOD AND APPARATUS


**Inventors:**

John Launchbury
Thomas Nordin
Mark Tullsen
William Bradley Martin



**Attorney Docket Number**
065335-137122

**Express Mail Label Number: EV422688494US**
**Date of Deposit: April 16, 2004**



*Schwabe, Williamson & Wyatt*
*1211 SW Fifth Avenue, Suites 1600-1900*
*Portland, OR 97204-3795*
*phone 503.222-9981*
*fax 503.796.2900*

# AUTOMATED PERMUTATION METHOD AND APPARATUS

## FIELD OF THE INVENTION

**[0001]** The present invention relates in general to the field of data processing, and more particularly to the modeling of permutations for programmable devices.

## BACKGROUND OF THE INVENTION

**[0002]** Cryptography involves the enciphering and deciphering of messages in secret code or cipher. Cryptography may deal with all aspects of secure messaging, authentication, digital signatures, and the like. The processing involved in such activities may involve complicated mathematical calculations. These calculations are the implementation of cryptographic algorithms. Such calculations have been implemented in software. Software implementations have advantages such as being relatively easy to update. This may be e.g. effectuated by downloading new software to the platform (i.e. hardware) running these algorithms.

**[0003]** Implementing algorithms in software, however, may have several shortcomings. Software implementations of algorithms typically are not able to process information as quickly has hardware implementations. Thus, in speed sensitive applications, frequently specialized hardware is utilized to perform at least a portion of the cryptography related functions.

**[0004]** One method of taking advantage of the speed of the hardware while maintaining the flexibility of software is to utilize programmable logic devices that are capable of being reprogrammed. In a hardware implementation utilizing such logic devices, flexibility may be maintained by the ability to reprogram logic to provide for

different implementations as needed. However, by still providing for the operation in hardware, quicker implementations may be achieved as compared to software only implementations.

**[0005]** Whether a cryptographic application is implemented in hardware or software, testing, configuration and/or verification have to be performed prior to its deployment. As described earlier, possible arrangements from successive permutations can become very large, very fast, thus make such testing, configuration, and verification extremely difficult.

## BRIEF DESCRIPTION OF THE FIGURES

**[0006]** Embodiments of the present invention will be described referencing the accompanying drawings in which like references denote similar elements, and in which:

**[0007]** FIG. 1 illustrates a system for automatically generating configurations files for use in configuring programmable devices on cryptography processors, in accordance with one embodiment.

**[0008]** FIGs. 2A-2C illustrate block diagrams of various aspects of a crypto processor in accordance with one embodiment.

**[0009]** FIGs. 3A-3B illustrates an "into" operation, in accordance with one embodiment, given two permutations.

**[0010]** FIGs. 4A-4B illustrate two permutations permuting inputs to outputs.

**[0011]** FIG. 5 illustrates a permutation of eight inputs to eight outputs and the generation of a subset from the outputs.

**[0012]** FIG. 6 illustrates a left rotation of a permutation by three positions.

**[0013]** FIG. 7 illustrates an inversion of a permutation, in accordance with one embodiment.

**[0014]** FIG. 8 illustrates the operation of a 'pad' function on a permutation, in accordance with one embodiment.

**[0015]** FIG 9 illustrates an example configuration file for one permutation for a permutation unit, in accordance with one embodiment.

**[0016]** FIG. 10 illustrates an expanded permutation definition for the definition illustrated in the example of FIG. 9.

**[0017]** FIG. 11 illustrates a portion of a configuration vector generated by the permutation processing engine, in accordance with one embodiment.

**[0018]** FIG. 12 illustrates a block diagram of a system capable of functioning as a permutation processing engine, in accordance with one embodiment.

## DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

**[0019]** Embodiments of the present invention include automated permutation methods and apparatuses. The methods and apparatuses have particular application to generating configuration files for programmable devices within a cryptographic processor, and for ease of understanding, will be primarily described in this context. However, the invention is not so limited, and may be practiced in other contexts.

**[0020]** Various aspects of illustrative embodiments of the invention will be described using terms commonly employed by those skilled in the art to convey the substance of their work to others skilled in the art. However, it will be apparent to those skilled in the art that the present invention may be practiced with only some of the described aspects. For purposes of explanation, specific numbers, materials, and configurations are set forth in order to provide a thorough understanding of the illustrative embodiments. However, it will be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well-known features are omitted or simplified in order not to obscure the illustrative embodiments.

**[0021]** The phrase "in one embodiment", "in accordance with one embodiment" and "in the embodiment" are used repeatedly. These phrases generally do not refer to the same embodiment; however, they may. The terms "comprising", "having" and "including" are synonymous, unless the context dictates otherwise.

**[0022]** FIG. 1 illustrates a system for automatically generating configurations files for use in configuring programmable devices on cryptography processors, in accordance with one embodiment. In the embodiment illustrated a permutation description file **110** is provided to a permutation generation engine **120**. Permutations may be utilized to

define the mapping of inputs to a logic device to outputs of the logic device. Optionally, a permutation library **130** is referenced to provide access to preexisting permutations. Examples of preexisting permutations may be Data Encryption Standard (DES) permutations. The permutation engine is utilized to generate one or more permutation unit configuration files **140** for configuring one or more programmable devices on a cryptography processor.

**[0023]** FIG. 2A illustrates a programmable cryptography engine **210**. FIG. 2B illustrates a block diagram of a portion of a programmable cryptography engine **210**. Various portions of the programmable cryptography engine **210**, such as permutation unit **220** and s-box unit **230**, may be programmed.

**[0024]** FIG. 2C illustrates inputs and outputs for a permutation unit **220**, in accordance with one embodiment. As illustrated in FIG. 2C, *4n* inputs **242-248** are provided to input of the permutation unit **220**. The inputs **242-248** are divided into four sets of *n* inputs and identified as IN1-IN4. The permutation unit **220** provides *4n* outputs **252-258** divided into four sets of *n* outputs and identified as O1-O4 **252-258**. In one embodiment, permutation unit **220** may be able to map any input to any output. In addition, permutation unit **220** may have the ability to map a single input to multiple outputs.

**[0025]** The permutation unit **220** may be programmed utilizing information regarding the mapping (i.e. permuting) of input bits to output bits. The generation of the bit level mapping information is typically performed manually. That is, a user may manually enumerate the mapping of one or more of the inputs **242-248** to one or more of the outputs **252-258**. Thus, up to *4n* mappings may be entered by the user. In the case

that a permutation unit **220** allows for multiple configurations, *m*, the user may need to manually provide 4n * m mappings.

**[0026]** Such a manual provision of mappings may have several disadvantages. First, the process may be error prone. This results from the manual provision of large numbers of bit patterns. For example, in the embodiment illustrated, the permutation unit may have multiple configurations of mapping 4n inputs and 4n outputs. For a permutation unit supporting eight configurations, this may result in the need to manually enter 4n*8 permutations. This can create large, manually entered configuration files thus creating the potential for error prone entry of configuration information. Second, when a large amount of data is manually entered into configuration files, it may be difficult to trace back the information in the configuration to an original design. Third, such a provision of manual information is time consuming.

**[0027]** As described below, the process illustrated in FIG. 1 may be utilized to receive higher level descriptions of desired permutations and generate the lower level information utilized to program the cryptography engines. By utilizing higher level descriptions to describe the desired programming of the cryptography engines, it is easier to maintain the source files utilized to generate the programming information. It also is easier to correlate the high level source to operations that are being preformed by the cryptography engines.

**[0028]** Permutations, as described herein, involve taking a set of inputs and permuting them to a set of outputs. For purposes of explanation, permutations can be specified as a list; with the list being an ordered list of the output of a permutation. The values specified for the outputs correspondingly identify the input sources to be permuted

(coupled) to the outputs (which are implicitly specified by the positions to the specified values). For example, a 4 bit rotation to the right may be illustrated as:

[4, 1, 2, 3]

Thus the fourth input is permuted to the first output, the first input is permuted to the second output, and so forth.

**[0029]** A set of operations may be defined to provide interactions between permutations and permutation modifiers. In such a case, names or symbols may be defined for each of the operations in the set. The names are illustrative of the operation being performed. However, the names utilized herein should in no way limited the implementations of embodiments of the invention. In addition, while the specific examples are provided utilizing small, e.g. several bit, permutations, it will be appreciated that the operations may apply to larger permutations.

**[0030]** One operator utilized to produce permutations is one that pipes the output of one permutation "into" the input of another. In such a case, the permutation modifier may be another permutation. For example, FIGs. 3A-3B illustrates an "into" operation, in accordance with one embodiment, given two permutations **310 320**:

[2, 4, 2, 3] and [4, 1, 2, 3]

The first permutation permutes the second input to the first and third output, the fourth input to the second output and the third input to the fourth, as illustrated **315**. The second permutation permutes the inputs to that second permutation as illustrated **325**. The composite permutation **335** of the first **310** and second **320** permutation, that is, by

inputting into a second permutation the output of a first permutation, is illustrated in FIG 3B.

**[0031]** Note that the number of inputs for a permutation does not necessary define the number of outputs. Illustrated in FIGs. 4A-4B are two permutations permuting inputs to outputs. FIG 4A illustrates **415** a [2, 3, 8, 5] permutation **410** of eight inputs to four outputs. Thus, there are a larger number of inputs than there are outputs. Similarly FIG 4B illustrates **425** a [7, 3, 1, 6] permutation **420** of eight inputs to four outputs. These two permutations may be "concatenated" (represented by the ++ operation) or joined side by side to form a single, eight input to eight output permutations. FIG 4C illustrates **435** an example of a composite permutation **430** based on the concatenated permutations of FIGs 4A and 4B.

**[0032]** A "select" operation may be performed on a permutation. The select operation may result in a subset of the permutated outputs being selected. FIG 5 illustrates **515** a permutation of eight inputs to eight outputs **510**. Utilizing the permutation illustrated in FIG. 5, a selection can be made of a subset of the outputs of the permutation. Shown below, a permutation modifier [3..6] is provided. In the embodiment illustrated, the permutation modifier implies that outputs 3 through 6 of the eight outputs of the permutation are to be selected:

[2, 3, 8, 5, 7, 3, 1, 6] 'select' [3..6]

As a result of the "select" operation, a resulting permutation of [8, 5, 7, 3] **520** may be generated.

**[0033]** Another set of possible operations on permutations are operations involving the rotation of a permutation. FIG. 6 illustrates **615** a "rotate left" by three positions permutation **610**. Thus, a permutation modifier of "3" may be provided:

[2, 3, 8, 5, 7, 3, 1, 6] <<< 3

where "<<<" is illustrative of a "rotate left" operation. Eight inputs may be permutated to outputs as illustrated in FIG. 6. The resulting permutation, [2, 3, 8, 5, 7, 3, 1, 6], may then be left shifted as illustrated **620**. In a similar manner, right rotations (i.e. "rotate right") of permutations may be specified. The right rotations perform the shifting of the output of a permutation to the right.

**[0034]** FIG. 7 illustrates an inversion of a permutation, in accordance with one embodiment. As with previous examples, the inputs are permuted to the outputs as specified **710**. From this permutation **730** an "inverse" permutation **740** is obtained. The inverse permutation **740** is defined as a permutation where the values in the inverse permutation **740** are the output position numbers in the permutation **730** of the corresponding output position of the inverse permutation **740**. For example, the value in the first position (1) **752** of the inverse permutation **740** is the corresponding output position number **756** in the permutation **730** of the input number 1 **754**. Similarly, the value in the second position (2) **762** in the inverse permutation **740** is the corresponding output position number **766** in the output permutation **730** of the input number 2 **764**. In this manner, the inverse permutation may be determined. In certain cases, the output position number may not exist in the output permutation. For example, input 4 is not a value in the permutation **730**. Thus, the corresponding output position **772** of the inverse permutation reflects this with a 1 in the output permutation **740**.

**[0035]** FIG. 8 illustrates the operation of a "pad" function on a permutation, in accordance with one embodiment. With certain permutations, the permuting of inputs to outputs may result in a permutation that has less entries than a desired width. The output of the permutation **815** in FIG. 8 illustrates a six position permutation [2, 3, 8, 5, 7, 3] **810**. However, it may be desired to have an eight position permutation. Thus, a 'pad' operator may be applied to the six position **842** permutation to obtain an eight position permutation **844**. The 'pad' operator may have a permutation modifier that specifies the number of positions in the resulting permutation. The padding may be accomplished by adding additional positions to the left of the six entries to form an eight position permutation [1, 1, 2, 3, 8, 5, 7, 3]. In alternative embodiments, a pad operator may operate to add additional positions to the right of the six positions. Additionally, while the illustration shows mapping of the first input, e.g. 1, to the padded outputs, in other embodiments other inputs may be used to pad the outputs.

**[0036]** FIG 9 illustrates an example configuration file for one permutation for a permutation unit, in accordance with one embodiment. Thus, a configuration number for the defined configuration is provided **910**. In various configurations, the number of configurations supported by the permutation unit varies. For example, in one embodiment, the permutation unit may contain up to 16 configurations. In the example illustrated, four *n* bit outputs O1-O4 **922-924** are defined. In addition, three working permutations are defined, *desE* **930**, *initialPerm* **932** and *expansion* **934**. *desE* **930** is a permutation defined in the configuration file. *initialPerm* **932** and *expansion* **934** are defined utilizing the operators discussed above and additional permutations. This includes a reference to *desIP* **940**, a permutation in a permutation library. Thus the

definition *desIP* **940** is not present in the configuration file but may be read from the permutation library by the permutation generation unit.

**[0037]** FIG. 10 illustrates an expanded permutation definition for the definition illustrated in the example of FIG. 9. That is, utilizing the functions discussed above, in combination with the values defined in the configuration file as well as the permutation library, expanded permutation definitions may be defined. For example, assuming that *in1* - *in4* are defined as in FIG. 2C, then output O3 is mapped to the inputs for *in1* which are inputs 1-16. The remaining outputs results from operations as discussed above. FIG. 11 illustrates a portion of a configuration vector generated by the permutation processing engine, in accordance with one embodiment. In the embodiment illustrated, the output illustrates detailed mappings of individual inputs to outputs. The outputs may be in an appropriate condition as to be compliant with programming software for the destined permutation. For example, in one embodiment, the configuration vector may be compliant with a toolkit utilized to compile the configuration vector into a final software image.

**[0038]** FIG. 12 illustrates a block diagram of a system **1200** capable of functioning as a permutation processing engine, in accordance with one embodiment. As shown, the system **1200** includes a processor **1210** and temporary memory **1220**, such as SDRAM and DRAM, on high-speed bus **1205**. High-speed bus is connected through bus bridge **1230** to input/output (I/O) bus **1215**. I/O bus **1215** connects permanent memory **1240**, such as flash devices and fixed disk device, networking interface **1260** and I/O devices **1250** to each other and bus bridge **1230**.

**[0039]** In the embodiment illustrated, permanent memory may be utilized to store the permutation library and permutation description file. The permutation description file may be structured as functional language statements in a functional language such as Haskell. Additionally, the permutation library may contain library files structured with functional language elements as well. Permanent memory may also comprise a Haskell execution environment for execution by the processor. In such a case the execution of the permutation description file, possibly along with Haskell portions of the permutation library, by the processor may result in the generation of the permutation unit configuration file.

**[0040]** Thus, it can be seen from the above descriptions, a novel method for modeling permutation operations, having particular application to the generating of configuration files for programmable devices on cryptography devices has been described. While the present invention has been described in terms of the foregoing embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described. The present invention can be practiced with modification and alteration within the spirit and scope of the appended claims.

**[0041]** Thus, the description is to be regarded as illustrative instead of restrictive on the present invention.